9143

#2
T.D.
01/27/03

THE AEROSPACE CORPORATION

Cnli0029427.doc
January 18, 2002

Confirmatory License

Application For:	Cooperative Adaptive Web Caching Routing and Forwarding Web Content Data Requesting Method	
Inventor(s):	Bartlett Scott Hudson Michel	
Serial No.:	09/810,303	Contract No.: F04701-00-C-0009
Filing Date:	03/16/01	Contractor: The Aerospace Corporation

The invention identified above is a "Subject Invention" under Patent Rights clause FAR 52.227, which is included in Contract No. F04701-00-C-0009, with the Department of the Air Force. This document confirms the rights acquired by the Government through the referenced clause, including a paid-up license in this invention, corresponding patent applications and in any resulting patents.

The Government is hereby granted an irrevocable power to inspect and make copies of the above-identified patent application.

Signed this 23 day of January 2002

Contractor: The Aerospace Corporation

By: _____
Gordon J. Louttit
Senior Vice President, General Counsel and Secretary

Business Address: P.O. Box 92957, M1/040
Los Angeles, CA 90009

INVENTION DISCLOSURE

00-29

IN DISCLOSING THE INVENTION

1. Title: Provide a brief title for the invention.
2. Technology: Indicate which Patent Review Committee should review this disclosure:
   ☐ Electronics ☒ Systems ☐ Physics ☐ Applied Mechanics
3. Abstract: Write a short paragraph discussing the invention, its components, and advantages.
4. Background: If you know of any highly relevant prior art, discuss the prior art's problems and methods without referring to the invention. Attach copies.
5. Invention Summary: Describe basic components of the invention and how the invention solves those problems.
6. Invention Description: Describe in detail the invention and its components while referring to elements shown in attached drawing(s).
7. Prior Disclosures: List and attach copies of all disclosures, public or private, of the invention made outside of The Aerospace Corporation.
8. Exploitation: List features of the invention that would give it a competitive advantage in the marketplace, if any, and discuss its potential uses and specific markets.
9. Funding: Was this invention government funded?
10. Execution: Sign the invention disclosure and have it witnessed by others who understand the invention. Send the invention disclosure to the Office of the General Counsel, Attn: Technical Counsel, M1/040.

Title: Application-level Routing and Forwarding

Abstract:

Application-level Routing and Forwarding (ALRF) is a mechanism for adapting the well-understood model of Internet routing and forwarding to the application layer (layer 4 [IETF model] or layer 7 [ISO model]). Normally, routing and forwarding are operations associated with Internet routers and the network layer (layer 2 [IETF model] or layer 3 [ISO model]). A specific instance of ALRF was developed for the *Adative Web Caching* project at UCLA, where *web caches* performed routing and forwarding of users' *World Wide Web* requests. A *web cache* is an intermediary device placed between a client (web browser) and a *WWW* server that replicates content in order to off-load request traffic from the *WWW* server and the underlying IP network. A web cache's fundamental operation is very straightforward: it receives a request for specific content identified by the content's *Universal Resource Locator* (URL) and checks whether or not the URL's content exists in the cache's *backing store*. The URL's content is immediately sent to the requester when it is present in the cache's backing store. Otherwise, the cache either relays the request to another cache or retrieves the content directly from the URL's *WWW* server. ALRF-WC provides a web cache with a low-cost decision process to determine whether to retrieve a requested URL's contents from its web server or send the request to an adjacent web server where the URL's contents are likely to exist.

Background:

Web caching is widely recognized in the Internet community as an important function that improves a browser's request retrieval time. Several companies are profiting in this business sector, such as *Network Appliances (NASD:NTAP)*, *Inktomi (NASD:INKT)* and *Akamai Technologies (NASD:AKAM)*. This research area was first proposed in the DARPA-funded *Harvest* research project with the *Harvest Internet Cache* [1], which gave rise to the *Squid* web cache system [3]. The *Harvest Internet Cache* was the foundation for what is currently offered as *Network Appliances' NetCache* product.

The basic principle behind a caching system, web-related or not, is that there exist intermediaries between a data source and a data sink in order to reduce the transfer cost between the source and the sink. it was observed by computer hardware designers that a small amount of intermediate memory (*cache memory*) inserted between the central processor and the main memory reduced both access and transfer time because programs exhibit *locality of reference*, i.e. a small group of instructions and data are repeatedly accessed.

In the *World Wide Web*, it was also observed that certain web pages and graphic files are repeatedly accessed. Copious examples abound: consider the home page each user sees when they first start their browsers such as the *Yahoo* or the *MSN* portals. Thus, if a device is inserted between a user's browser and the *WWW* servers that the user accesses, the transfer cost of a group of users' requests could be reduced, assuming this device stores local copies of frequently request web pages. If a user's request exists in the web cache, the contents are immediately sent back to the user's browser without communicating with the re-

quest's origin *WWW* server. Many methods can be used to determine what content is "hot", ranging from using the law of large numbers and least recently used (LRU) techniques to data mining the web cache's *backing store*. "Backing store" is a generic term for a storage mechanism where a web cache places copies of frequently requested content.

It was proposed in [1] that web cache devices cooperate with each other, where aggregate web caching system appears larger than just a single web cache device. The relationships between the web cache devices are organized into interconnected, hierarchical peer groups. Each web cache is manually configured to communicate with a list of peers and parents. When the web cache receives a request and the request's contents are not in its backing store, it first sends the request to its peers and awaits a response. If no response is received from the peer group, the web cache sends the request to its parent and awaits a response. The request percolates through peer groups and parents until a root cache peer group (i.e. caches without parents) is reached. Root web caches always fetch the request directly from the original *WWW* server when the request's contents are not found in their respective backing stores. The request's contents travel back down from the cache that satisfied the request (root or parent) to the lowest level, leaf cache.

The static network topology between web caches and the additional request latency it introduces have been the study of various papers, such as [4] and [5]. It was proposed in [6] that a *Universal Resource Locator's* (URL's) signature could be used in a Bloom filter to determine which web cache in a peer group, if any, had the URL's contents with a high degree of precision. This requires each web cache to periodically compute the Bloom filters for all URLs stored in its backing store and send the filter's bit string to its peers. This reduces or eliminates the time required to poll a web cache's peers. However, it does not indicate which parent is desirable to send the URL next. This demonstrates the need for a better mechanism or process in a web caching system to locate where a URL's content is stored or to select a path through a network of cooperating web caches along which the content is likely encountered.

The concept of *Application-level Routing and Forwarding for Web Caches* originated as part of my Ph.D. research in the *Adaptive Web Caching* (AWC) project [7] at UCLA. The project was composed of two parts: the *Cache Group Management Protocol* (CGMP) and the *Content Routing Protocol* (CRP). My contribution was solely to CRP and I am the primary author of publication [2].

The basic premise behind AWC is rapid dissemination of frequently accessed Web content through the cache network based on measured demand. This is accomplished in two phases: CGMP constructs the interconnected cache group network, and CRP propagates where cached content is located using routing protocols in order to subsequently relay requests toward a cache where its contents are likely to be found.

The *Cache Group Management Protocol* constructs an interconnected cache mesh by executing an algorithm that causes each cache to self-organize and to join cache groups containing neighboring caches. CGMP encourages the caches to join more than one group, resulting in interconnected groups. Web servers also participate in the AWC caching system's cache groups via CGMP but only join a single cache group. The key observation is that these interconnected groups resemble a network built at the application level.

The *Content Routing Protocol* is implemented using two different routing protocols: a *source location* routing protocol and *local content state* routing protocol. The *source location* routing protocol is based on the *distance-vector* style of routing protocol used in the Internet [8], and disseminates Web server prefixes such as "http://www.aero.org/*" representing the Web addresses corresponding to the *aero.org* Web server. Source location routing updates originate at the Web server and percolate through the cache groups. This creates a tree rooted at the originating web server and terminating at the web caches connected to web clients and browsers. The *local content state* routing protocol identifies content (web pages, graphic images, content style sheets, etc.) currently stored within a single cache group. Since the amount of content stored per web cache is potentially quite large, an *incremental hashing* technique is used to represent each individual content's URL.

Hashing is the process by which a hashing function transforms a variable length block of data into a fixed length numeric code. Incremental hashing is a hashing process that creates a series of numeric codes based on incrementally hashing successive pieces of the input block of data. Hashing is imprecise because different input data blocks may transform to the same numeric code. The probability that the hashing function maps the different input data blocks to the numeric codes is known as its *collision factor*. Low collision factors are desirable. A 19-bit *cyclical redundancy check* function is used in the local content routing protocol, and has a 2^{-19} (1.907×10^{-6}) collision rate.

Each routing update from either the source location or local content state protocols updates each web cache's *forwarding table*. The forwarding table contains associations between both the Web server prefixes and incremental hash codes and the next-hop IP address of a web cache. For web server prefixes, the next-hop web cache is the next node the tree that leads the indicated web server at the tree's root. For incremental hash codes, the next-hop web cache is the cache where the requested URL's contents are stored with 99.999809% certainty.

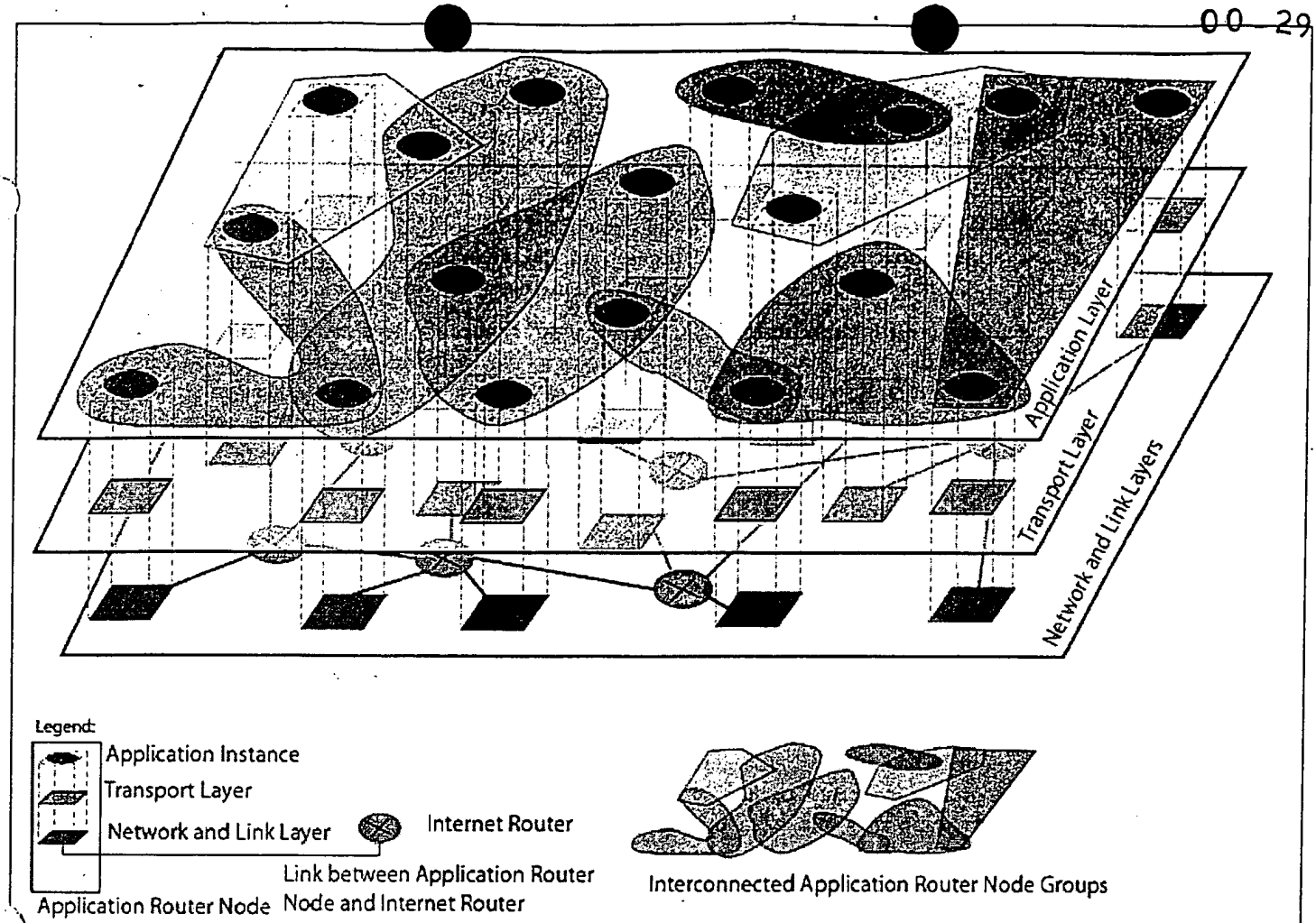

Figure 1: Diagram of an Application Level Routing System

For each request received, the web cache first converts the request's URL into a list of components and creates the sequence of incremental hash codes. The forwarding table is probed to see if the entire hash code sequence exists. If the hash code sequence does not exist, the forwarding table is probed again for the longest matching Web server prefix. When either a matching hash code sequence or web server prefix is found, the request is forwarded to the indicated next-hop web cache. Otherwise, the request is fetched directly from the URL's designated web server.

The process of disseminating or propagating the web server's prefix and the incremental hash code sequences and associating them with a web cache's next-hop IP address is similar to the operation of an Internet router. Internet routers normally operate at the *Network Layer* (see figure 1) and propagate the IP network addresses. Internet routers maintain a forwarding table that stores the associations between the IP network addresses and the next-hop router's IP address leading toward an IP network. The principle differences between the two systems are:

- The AWC web caches route and forward at the application layer of the network protocol stack whereas IP routers operate at the network layer
- Web caches propagate prefixes and names whereas IP routers propagate IP network addresses in routing protocol updates

The focus of my PhD dissertation is a generalization of my contribution to AWC, a toolkit that provides general application-level routing and forwarding services. This is described in my PhD prospectus, which is available upon request.

Description:

Application-Level Routing and Forwarding: A General Description

A diagram of a general application-level routing and forwarding system is shown in figure 1. It consists of a collection of *application router nodes*. An application router node is a computer system that executes an *application instance* participating an *application-level network*. An application instance is a program that sends messages to other application-level router nodes. An application-

level network is a network where the node topology and interconnection between nodes is created at the application layer of the network stack, layer 4 in the *Internet Engineering Task Force's* (IETF) network model or layer 7 in the ISO network model. This is in contrast with the usual definition of inter-networking, which is typically understood to operate at the network layer layer 2 in the IETF model or layer 3 in the ISO model. Messages sent between application-level router nodes originate at t. application layer in an *application instance*, transit through a node's transport, network, and link layers, transit through one or more inter-networking router nodes, and are received at a destination node. Generally speaking, technologies such as Ethernet, cable modems, xDSL and the like, link application-level router nodes to inter-networking routers.

Before any message can be sent between application-level router nodes, the application-level network must be constructed by a *topology construction protocol.* This topology construction protocol is responsible for creating the application-level router node groups (the colored regions in figure 1). Interconnection between node groups occurs when an application-level router node's topology construction protocol causes the node to participate in more than one group (the overlapping colored regions in figure 1). It is assumed that a topology construction protocol exists that creates the interconnected groups, although none is specified in this invention description. The topology construction protocol may be as simple as pre-defined (static) interconnections between nodes or a more complicated process that creates dynamic interconnections, for example, self-organizing or emergent-behavior protocols.

An application-level router node processes two classes of messages: control and data. Control messages consist of *routing protocol* updates that propagate parts of a *name space* specific to the application. Data messages are requests that the node either forwards, or relays, to another node or performs some computation upon, usually sending a reply or status message back to the request's originator. Control messages change the state of each node's *forwarding table*, which in turn controls where data messages are relayed. A forwarding table contains associations between the application's name space and the network address or addresses of other application-level router nodes. A single association is referred to as a *forwarding entry.* Each data message has contained within it a specific name, which is matched against the names currently present in the forwarding table, identifying a forwarding entry. If a forwarding entry is located, the data message is relayed to the network addresses of the application-level router nodes. Otherwise, the node's application instance invokes a *fallback behavior* to process the message.

AWC Caches: Basic Operation

In the AWC system, web caches are the application-level router nodes. The application instance is the program that partu. pates in the AWC system. Each application instance receives HTTP protocol requests and determines whether or not the *universal resource locator* (URL) contained in the HTTP request is present in the web cache's backing store or memory. If the URL's contents exist in the web cache's memory or backing store, its contents are immediately sent to the HTTP request's originator (a web cache or user's browser). If the URL's contents do not exist, the AWC web cache decomposes the URL, performs a longest-matching prefix algorithm with the URL and its URL forwarding table. If a URL forwarding entry is found, the received HTTP request is relayed to the web cache indicated in the URL forwarding entry. Otherwise, the URL's contents are fetched directly from the web server indicated in the URL.

When a web cache receives a requested URL's content, it commits the content to its own backing store, thereby replicating the URL elsewhere in the AWC web cache system. URL content does not persist forever in a web cache's backing store, but are expired periodically to make room for newer or more heavily demanded URL content.

The topology construction protocol is implemented by the *Cache Group Management Protocol* (CGMP) software component. CGMP creates overlapping web cache groups that contain both AWC web caches and web servers. Only web caches are allowed to be part of more than one web cache group, thus, only web caches perform HTTP request forwarding. CGMP is based on a self-organization algorithm.

Hierarchical URL Decomposition: The Name Space

The name space used by AWC web caches is formed from hierarchical URL decomposition. Each URL is parsed apart into a sequence of components identified by the URL's scheme, host name (network locator), and path. The host name is further decomposed in reverse, expressing the hierarchy present in the Internet's *Domain Name System.* For example, the URL http://info.aero.org/tai/courses/2000/fall/index.html is decomposed into the sequence (*http, org, info, tai, courses, 2000, fall, index.html*). A collection of decomposed URLs is easily organized as a tree, as shown in figure 2, (taken from [2].) The URLs used to construct this tree were http://www.fribbles.org/index.html, http://www.fribbles.org/subdir1/index.html, http://www.fribbles.org/subdir1/foobar.gif, and http://www.yahoo.com/*. The last URL contains the "*" wildcard character, ... noting a URL fragment. URL fragments are a compact notation used to represent subtrees, in this case, all of the URLs contained in the www.yahoo.com web server.

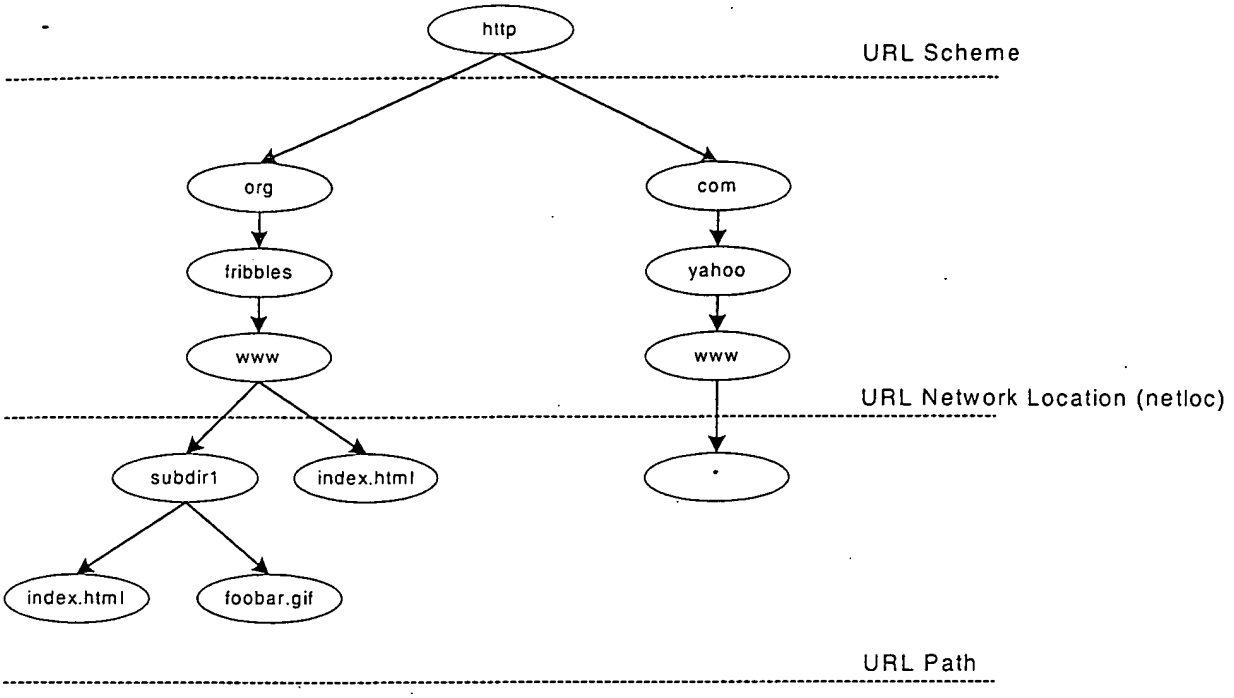

Figure 2: Example Hierarchical URL Decomposition Tree

The URL Forwarding Table

The URL forwarding table is a data structure used to represent associations between a URL or URL fragment and the IP address of the next-hop web cache, which either leads toward the web server or contains a requested URL's content. The forwarding table's structure is based on the tree of decomposed URLs, however, for performance reasons specified in [2], an *incremental hashing* technique is used instead. A hash table is a simple table structure where the table's indices correspond to the result of a hash function. Each index in the table is commonly referred to as a hash bucket. The hash function converts an input, usually a string of characters, into a number from a restricted range. In this application, a special hash function provides a sequence of related hash codes from the list of decomposed URL elements. For example, the URL http://www.fribbles.org/index.html produces the incremental hash code sequence using the hash function H(x, string):

$$h_0 = H(0, \text{"http"})$$

$$h_1 = H(h_0, \text{"org"})$$

$$h_2 = H(h_1, \text{"fribbles"})$$

$$h_3 = H(h_2, \text{"www"})$$

$$h_4 = H(h_3, \text{"index.html"})$$

Each successive hash code is built from the previous code's value and the next decomposed URL element. In the AWC implementation, a 19-bit *cyclical redundancy check* (CRC) generator was used with the polynomial (0xf10eb):

$$x^{19}+x^{18}+x^{17}+x^{16}+x^{12}+x^{7}+x^{6}+x^{5}+x^{3}+x^{1}+x^{0}$$

This gives the size of the resulting hash table as 2^{20}-1 (1048575) elements. The other three important storage characteristics of the URL forwarding table are:

- Relationships between decomposed URL elements are kept by pointers to the preceding decomposed URL element. This preserves a tree-like structure.
- "Placeholders" may be inserted that correspond to a *hash chain* sequence learned from the local content state routing protocol (vide infra). A hash chain represents a single, complete URL. Only one placeholder will exist per
- Only the last decomposed URL element and final placeholder in a hash chain sequence point to the next-hop web cache's IP address (and other useful information).

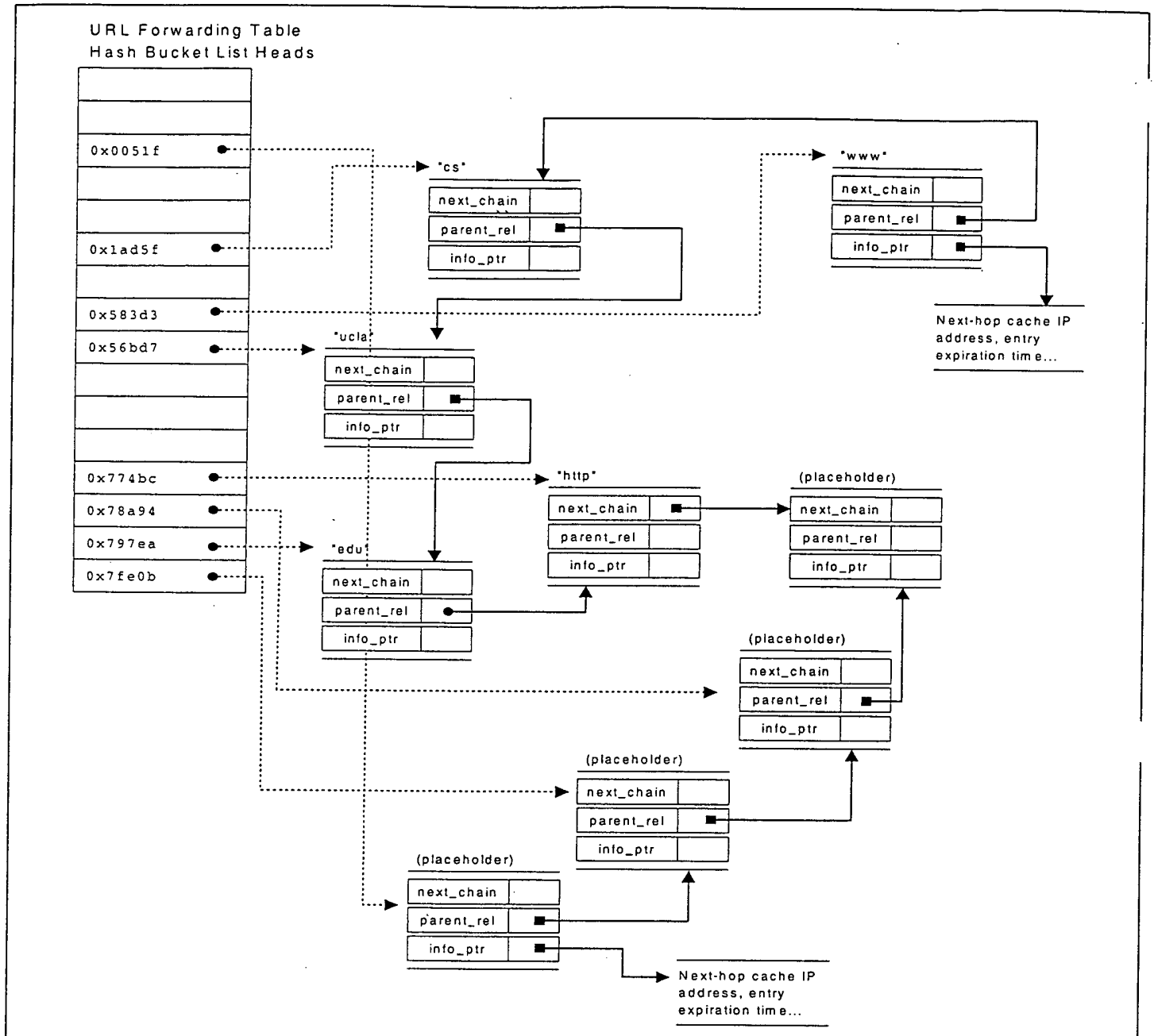

Figure 3: Example URL forwarding table

A sample URL forwarding table is shown in figure 3. In this figure, the URL http://www.cs.ucla.edu/ and a hash chain sequence (0x774bc, 0x78a94, 0x7fe0b, 0x0051f) have been inserted. The structures to which the final elements (the "www" element for http://www.cs.ucla.edu/ decomposed URL and the 0x0051f hash chain element) point to are managed separately from the URL forwarding table. Thus, it is straightforward to locate expired forwarding table entries after routing updates have been processed. Also, note in figure 3, the hash buckets are explicitly labeled in order to save space. In reality, the hash code's bucket is located by indexing into a 1048575 element pointer array.

A forwarding entry is located by taking the decomposed URL and computing each of the incremental hash codes for ea... decomposed element. Conceptually, three passes are made through the forwarding table. The first pass identifies if a hash chain sequence exists for the requested URL, and if so, returns the IP address of the next-hop cache where the URL's content

will be found. The second pass identifies whether the entire text form of the decomposed URL exists, and if so, returns the IP address of the next-hop cache along the path leading to the web server. The third and final pass attempts to identify the longest matching URL fragment in the forwarding table and returns the IP address of the next-hop cache along the path leading to the web server if such exists. A more detailed description of the forwwarding algorithm follows, using the URL http://www.cs.ucla.edu/ as the requested URL. The URL decomposes into the elements "http", "edu", "ucla", "cs" and "www". The series of hash codes generated in hexadecimal notation are 0x774bc, 0x797ea, 0x56bd7, 0x1ad5f, and 0x583d3.

1. The first hash code's bucket is searched for a placeholder. If a placeholder is found, steps a – c are executed, otherwise step 2 is executed.
   a. The second hash code's bucket is examined for a placeholder and its parent pointer is examined to insure that the first hash code bucket's placeholder is the second hash code's parent. If no placeholder is found or no child-parent relationship exists between the second hash code bucket's placeholder and the first hash code bucket's placeholder, step 2 is executed.
   b. Step a is repeated for the third and remaining hash codes and their respective buckets until the list of hash codes is exhausted.
   c. If the final hash code is the child of the penultimate hash code and the final hash code buckets placeholder's info_ptr points to a next-hop IP address structure, then the forwarding algorithm returns the next-hop IP address structure. Otherwise step 2 is executed.

   In figure 3, the search starts at the bucket labeled 0x774bc, corresponding to the "http" URL scheme. Note that the algorithm will find the placeholder in this case. However, steps a – c will not find any placeholders in the hash bucket 0x797ea (the hash code generated by "http" and "edu") and consequently fails. However, if a requested URL had generated the incremental hash sequence 0x774bc, 0x78a94, 0x7fe0b, and 0x0051f instead, this portion of the algorithm would have returned the info_ptr structure pointed to by the placeholder in the 0x0051f hash bucket.
2. Starting with the first hash code, examine the first hash code's bucket list to find an element who's string matches the first decomposed URL element's string (i.e. "http"). If no match is found, return from this algorithm with an error indication. Otherwise, execute steps a - d below.
   a. The second hash code's bucket list is searched for an element who's string matches the second decomposed URL element's string (i.e. "com", "edu", etc.)
   b. If a matching element is found, the child-parent relationship is checked between the second element and the first element. If no matching element is found, step 3 is executed.
   c. If the second element is the child of the first element, repeat steps i and ii with the third and remaining hash codes until the hash code list is either exhausted or the child-parent relationship is invalid.
   d. If the hash code list is exhausted and the final hash bucket element's info_ptr points to a next-hop IP address structure, the forwarding algorithm exits returning the next-hop IP address structure. Otherwise the forwarding algorithm terminates with an error indication.

   In this part of the algorithm, the 0x774bc hash bucket is searched for an element labeled "http", which is found. Next, the 0x797ea hash bucket is searched for an element labeled "edu", which is also found. The "edu" parent_rel pointer is checked to see if it refers to the "http" element, which it does. The algorithm proceeds to locate the "ucla" element in the 0x56bd7 hash bucket and checks to see if its parent_rel pointer refers to the previous "edu" element. This process repeats for the "cs" and "ucla" elements in hash buckets 0x1ad5f and 0x583d3 hash buckets respectively. Since the hash code list is exhausted, the "ucla" info_ptr structure is returned by the forwarding algorithm at this point and step 3 would be skipped.
3. If the current hash code bucket does not contain an element matching the corresponding decomposed URL element or the element matching the decomposed URL is not a child of the previous hash code bucket's element matching the previous decomposed URL element:
   a. Compute the hash code using the previous hash code and the wildcard string "*", examining this new hash code's bucket for an element who's string matches the wildcard string.
   b. If a wildcard string element is found and it's info_ptr points to a next-hop IP address structure, the forwarding algorithm terminates returning the next-hop IP address structure.

c. Steps a – b are continued working backwards in the hash code list, e.g. from h_3 to h_2 to h_1 to h_0 if step 2 failed at hash code h_4. The forwarding algorithm terminates at this point with an error indication when the first hash code, h_0, is processed.

If step 2 had failed or the "ucla" element did not have a valid info_ptr, step 3 would have looked for the URL fragment http://cs.ucla.edu/* (decomposed as "http", "edu", "ucla", "cs", *), http://ucla.edu/* ("http", "edu", "ucla", "*"), http://edu/* ("http", "edu", "*"), and http://* ("http", "*"), in that order.

Source Location and Local Content State: The Routing Protocols

There are two basic control messages exchanged between AWC web caches: *source location* and *local content state* routing updates. The source location routing protocol is based on the principles for distance-vector protocols found in IETF Standard RFC 2453 [8]. Its purpose is to establish a collection of trees rooted at each web server, who's branches extend through the web caches groups and terminate at the "leaf" web caches connected to the user's browsers. This allows a request to transit from one cache to the next up the tree if the request's content is neither present in the receiving cache's backing store nor located within receiving cache's cache group(s). The local content state routing protocol operates within a single cache group. Its purpose is to inform the neighboring web caches what URLs' contents are currently stored in a given cache's backing store.

The source location routing updates are initiated by each web server on a periodic basis, e.g. once every 5 minutes. A web server creates a message containing a URL fragment with its host name, e.g. http://www.yahoo.com/*, and a distance metric initialized to the value one (1) to the web caches in its local web cache group. Each web cache that receives the source location routing protocol message extracts and decomposes the web server's URL fragment and examines its URL routing table. The routing table is a collection of URL fragments associated with a list of 3-tuples. The 3-tuple contains a sender's IP address, distance metric, and an expiration time. The web cache chooses the sender's IP address having the minimum distance metric and updates its URL forwarding table with an association between the decomposed URL fragment and the sender's IP address with the minimum distance metric. The web cache subsequently creates a new source location message containing all URL fragments present in its routing table with the minimum distance incremented by one. The new message is sent to the web cache's other web cache groups, excluding the web cache group from which it received the source location routing update. This process builds a shortest path tree rooted at the web servers through the web cache groups. A source location routing update process terminates when no new routing messages are created by any web cache. RFC 2453 [8] contains additional useful guidelines and refinements that should be observed, such as split-horizon processing, route expiration and garbage collection. "Split-horizon" processing is a refinement where route information is never sent back to its originator. The means that if some web cache *B* receives source location updates from some web cache group A, *B* never sends any URL fragments learned from *A* back to *A*.

The local content state routing update consists of a sequence of incremental hash codes produced from the URL's stored in a web cache's backing store, using the same 19-bit CRC function and URL decomposition algorithm. This assumes that the web cache's backing store can be organized in a tree structure as shown in figure 2 or can be enumerated or traversed in such that a tree structure is the result. This tree is traversed depth-first and in-order to produce a *hash chain* sequence, a list of <*depth, hash code*> pairs. *depth* indicates the depth in the tree where the hash code occurred. *hash code* is the incremental hash code computed along a traversed path from the root of the tree to an arbitrary node. The tree in figure 2 produces the following hash codes as it is traversed depth-first and in-order:

h_{00}= $H(0,\text{http})$	h_{10}= $H(h_{00}, \text{org})$	h_{20}= $H(h_{10}, \text{fribbles})$	h_{30}= $H(h_{20}, \text{www})$	h_{40}= $H(h_{30}, \text{subdir1})$	h_{50}= $H(h_{40}, \text{index.html})$
					h_{51}= $H(h_{40}, \text{foobar.gif})$
				h_{41}= $H(h_{30}, \text{index.html})$	
	h_{11}= $H(h_{00}, \text{com})$	h_{21}= $H(h_{11}, \text{yahoo})$	h_{31}= $H(h_{21}, \text{www})$	h_{42}= $H(h_{31}, *)$	

The hash chain sequence for figure 2's tree would be:

$$<0,h_{00}>,<1,h_{10}>,<2,h_{20}>,<3,h_{30}>,<4,h_{40}>,<5,h_{50}>,<5,h_{51}>,<4,h_{41}>,<1,h_{11}>,<2,h_{21}>,<3,h_{31}>,<4,h_{42}>$$

The hash chain sequence is sent to all of the cache groups of which the sending web cache is a member. A cache receiving a local contents state update message executes the following algorithm on each <*depth, hash code*> pair.

1. Insert a placeholder element in the hash bucket for *hash code.*
2. If *depth* is greater than 0, create a link to its parent placeholder element at *depth – 1*, the preceding depth.

3. If the current pair's depth is less than or equal to the preceding pair's depth, set the preceding placeholder element's info_ptr to a structure containing the IP address of the local content state routing update's sender.
4. If the current pair is the last pair, set its placeholder element's info_ptr to a structure containing the IP address of the local content state routing update's sender and terminate the algorithm.

The principle advantage of transmitting the URLs in this way is compression and performance, as explained and stated in [2].

What's Patentable?

This is my assessment (opinion) on what is patentable in this disclosure, modulo prior art documented in the USPTO's database:

- The concept of application-level routing and forwarding, in general, using an artificially constructed topology, routing protocols, and forwarding tables built and maintained at the application layer, independent of the underlying network infrastructure
- The local content state update protocol and its relationship with the URL forwarding table, how the generated list of hash codes are used to locate cached URL content, and the URL forwarding algorithm

The source location routing protocol is likely not patentable, due to its basis on IETF RFC 2453. However, its description or mention of a routing protocol, which operates and connects the web caches across the web cache groups will be needed in the patent filing.

The concept of *Adaptive Web Caching* is not patentable, since my work on URL routing and forwarding was a contribution to that research effort. The principal investigator for AWC is Lixia Zhang, Assistant Professor, UCLA Computer Science Department. She can be reached via e-mail at lixia@cs.ucla.edu.

Ultimately, it would be desirable to write the patent in such a way that application-level routing and forwarding in a general sense is covered, where my contribution to the AWC project is an example. I'd prefer not to go through another invention disclosure again with respect to my continuing PhD research efforts if I don't have to.

Commercial Exploitation

This concept appears to have considerable applicability and opens up an entirely new market in its own right. My current research thrust as a PhD student in the Aerospace PhD fellowship program is to produce a general programming toolkit for application developers so that they can take advantage of application-level routing and forwarding. One would use this toolkit in a variety of application areas where data or services are replicated or segmented across multiple computer systems. For example, in a distributed data fusion environment, one could imagine an user's application using application-level routing and forwarding to locate fused products and products in the process of being fused. The routing process makes the application aware of data or services provided by one or more service providers and the forwarding function directs the requests to the appropriate service provider. The level of appropriateness is determined by the routing protocol used, such as "closest" relative to some distance metric or security parameters contained in the routing protocol's update messages.

While Cisco might not be immediately interested in hardware product development, in the marketplace today, there exist "content switching devices" that examine World Wide Web HTTP protocol messages. A leading example of a content switching product line is Alteon Web Systems (http://www.alteonwebsystems.com). They provide the ability to direct WWW requests based on the request's source IP address and network, browser cookies contained in the request, or the request's referrer (i.e. click-through banner advertisement at eXcite!) However, these devices only operate on specific application traffic types, such as WWW requests. If adopted or licensed to commercial hardware vendors, a more generalized application-level and forwarding "switch" would provide the ability for Internet service providers to create a new class set of differentiated application services. Routing allows the Internet service provider to "engineer" their traffic, something they do today at the network level and not at the application level. Application traffic engineering would allow a customer to request and the Internet service provider to charge for differentiated treatment of name-to-IP-address resolution or content transcoding. These tools do not exist, in a general sense, today nor are they an integral part of applications. One would also expect software developers and operating system vendors to integrate application-level routing and forwarding into their systems because it manages much of the inter-entity communication and organization in a way that looks like a network. Application-level routing and forwarding also provides a means for adding new features to a distributed application without redeploying the entire application and infrastructure.

How was this funded?

Michael Campbell, Principal Director of CSTS, would have the records the Aerospace MOEI that I charged while I was developing this concept as part of my research here at Aerospace in the PhD fellowship program. At no time was I paid or supported by my (then) advisor at UCLA, Dr. Lixia Zhang. The same condition exists today with respect to my current advisor, Dr. Peter Reiher. Dr. Reiher also works as an Aerospace casual employee.

References:

[1] A. Chankhunthod, P. Danzig, C. Neerdaels, M.F. Schwartz, and K.J. Worrell, "A Hierarchical Internet Object Cache," Technical Report 95-611, Computer Science Department, University of Southern California, Los Angeles, California, March 1995.

[2] B. Scott Michel, Peter Reiher, Konstantinos Nikoloudakis, and Lixia Zhang. URL Forwarding and Compression in Adaptive Web Caching. In INFOCOM. IEEE, March 2000.

[3] National Laboratory for Applied Network Research, *Squid Internet Object Cache*, http://squid.nlanr.net/Squid/.

[4] N.G. Smith, "The UK national Web cache-the state of the art," *Computer Networks and ISDN Systems*, vol. 28, no. 7-11, pp. 1407–14, May 1996, (Fifth International World Wide Web Conference, Paris, France, 6-10 May 1996.).

[5] D. Neal, "The Harvest object cache in New Zealand.", *Computer Networks and ISDN Systems*, vol. 28, no. 7-11, pp. 1415–30, May 1996, (Fifth International World Wide Web Conference, Paris, France, 6-10 May 1996.).

[6] Li Fan, Pei Cao, Jussara Alameida, and Andrei Broder, "Summary Cache: A Scaleable Wide-Area Cache Sharing Protocol," in *ACM SIGCOMM'98*, 1998, pp. 245–265, Technical Report 1361, Computer Sciences Department, Univ. of Wisconsin-Madison, Feb 1998.

[7] S. Floyd, L. Zhang, and V. Jacobson, *Adaptive Web Caching*, http://irl.cs.ucla.edu/awc.html, DARPA-funded Research Project, May 1997.

[8] Malkin, G. "RIP Version 2", Internet RFC 2453, http://www.isi.edu/in-notes/rfc2453.txt.

Bartlett S. H. Michel PLEASE WRITE SIGNATURES AND DATES IN BLACK INK

INVENTOR	DATE	INVENTOR	DATE	INVENTOR	DATE
Bartlett S. H. Michel	20 Sep 00				
ORGANIZATION AND CCC		ORGANIZATION AND CCC		ORGANIZATION AND CCC	
CSRD (5812)					
WITNESSED, READ AND UNDERSTOOD BY:	DATE:	AND BY:			DATE: